Cross Domain Model Compression by Structurally Weight Sharing

Shangqian Gao[1], Cheng Deng[2], and Heng Huang*[1,3]

[1]Electrical and Computer Engineering Department, University of Pittsburgh, PA, USA
[2]School of Electronic Engineering, Xidian University, Xi'an, Shaanxi, China
[3]JD Digits

shg84@pitt.edu, chdeng.xd@gmail.com, heng.huang@pitt.edu

Abstract

Regular model compression methods focus on RGB input. While cross domain tasks demand more DNN models, each domain often needs its own model. Consequently, for such tasks, the storage cost, memory footprint and computation cost increase dramatically compared to single RGB input. Moreover, the distinct appearance and special structure in cross domain tasks make it difficult to directly apply regular compression methods on it. In this paper, thus, we propose a new robust cross domain model compression method. Specifically, the proposed method compress cross domain models by structurally weight sharing, which is achieved by regularizing the models with graph embedding at training time. Due to the channel wise weights sharing, the proposed method can reduce computation cost without specially designed algorithm. In the experiments, the proposed method achieves state-of-the-art results on two diverse tasks: action recognition and RGB-D scene recognition.

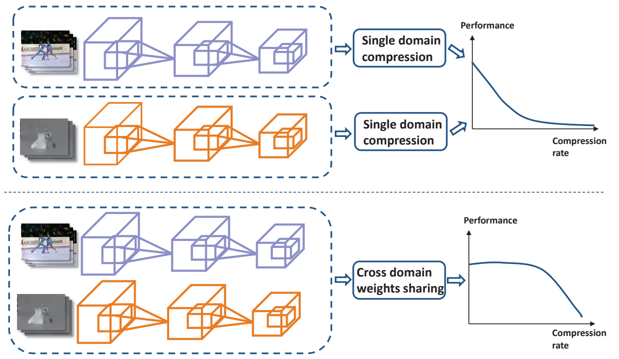

Single domain compression
Single domain compression
Performance
Compression rate
Cross domain weights sharing
Performance
Compression rate

Figure 1: A demonstration of difference between our method and single domain compression method. **Upper figure** shows that when it comes to cross domain compression, simply using regular compression method won't achieve satisfactory trade-off between performance and compression rate. **Lower figure** shows that sharing weights across domain can achieve good result.

1. Introduction

In recent years, Convolution Neural Networks (CNNs) have become very popular in many related fields, for instance, image classification [3, 23] , action recognition [37, 4] , self-driving cars [1], and so on. However, as CNNs go deeper and deeper [10, 14], the memory footprint and computational cost have increased dramatically, making it impractical to deploy on platform with limited resources such as mobile phone and embedded device. To resolve such problem, countless efforts have been made [8, 43, 7, 17]. These methods for CNN model compression can be separated to four categories: pruning [8], sparsity induced regularization [43], weight quantization [7], and low rank factorization [17].

Although compression techniques have been widely developed for RGB input. Cross domain applications are seldom considered for applying compression algorithms. Despite little attention is given on cross domain tasks, the memory cost and computation demand are even higher than single RGB domain. Popular cross domain applications like RGB-D scenes recognition [6], action recognition [37], cross domain retrieval [20, 25] etc, usually use two or more DNN models to collect domain specific information from different sources. Thus the storage cost, memory footprint and computation cost are at least two times higher than single RGB task. As a result, it is worthy to explore how to get compact models for cross domain tasks.

*Corresponding Author. S. Gao, H. Huang were partially supported by U.S. NSF IIS 1836945, IIS 1836938, DBI 1836866, IIS 1845666, IIS 1852606, IIS 1838627, IIS 1837956.

In cross domain tasks, the distinct spatial structure and appearance among different data sources often prohibit directly using mainstream compression methods. Indeed, when applied on cross domain tasks, mainstream compression methods have many drawbacks. Cross domain models are extremely sensitive to channel wise pruning. The hyperparameter search is more difficult for sparsity induced methods. Furthermore, mainstream compression methods can't utilize underlying cross domain relationships to achieve better compression rate.

To tackle above problems, we propose a new cross domain compression method which is robust to hyperparameter settings and can utilize cross domain relationships for better model compression. In the proposed method, the weights are structurally shared across domains. To achieve structured weight sharing, cross domain models are trained with graph embedding regularization. After training complete, the weights are clustered based on intermediate feature similarity graph. In the end, the cross domain models are fine-tuned to get final result.

The main contribution of this paper can be summarized to three aspects:

1. We identified the difficulties of cross domain compression when using regular sparsity induced methods as well as pruning algorithms.

2. We proposed a new method specially tailored for cross domain compression by using graph embedding as a constraint at training time. Proposed method is robust to hyperparameter tuning and it can naturally achieve computation cost reduction.

3. Proposed method can achieve the best results on two diverse tasks (action recognition and scene recognition) compared to other methods.

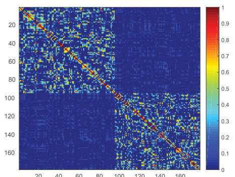
2. Related Works

The related work for this paper can be separated into two different perspectives, the first part is related to model compression, and the second part is about cross domain tasks.

2.1. Model Compression

Pruning and weight sharing methods are most related to our method. Thus, we mainly focused on these algorithms.

For weight sharing, there is a group of algorithms [33, 7, 16, 34, 50] studying how to clustering the scalar values of weights into several clusters. This kind of algorithms is also known as quantization. One of the earliest works [7] which combines quantization and hamming coding comes from this category. With weight quantization, the weights can be reduced to at most 1 bit binary value [15] from 32-bits float point numbers. Many works show that quantizing weights to 8-bits [49] often won't hurt the performance. A series of less popular approaches is about structured weight sharing. Unlike quantization, structured weight sharing focus on finding the structure level similarity across channels or filters. Learning to share [47] belongs to this category, which aims to find similarities across input channels by using a regularization term called group weighted order lasso (GrOWL) [31]. WSNet [21] tries to create a shared filter bank instead of finding similarities. In audio classification tasks, WSNet can achieve state-of-the-art result. Our method for cross domain compression is close related to these methods. Despite the similarity between our method and structured weight sharing methods, our method is a fully weight-sharing approach unlike Learning to Share, and the shared filter banks is learned during training, which is also different with the pre-designed filter bank in WSNet.


(a) visualization of weights correlation

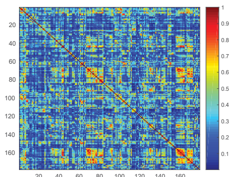

(b) visualization of features correlation

Figure 2: (a) is the weights correlation of cross domain MNIST experiment. (b) is the feature correlation of cross domain MNIST experiment. From (a), (b), we can see that model trained with GrOWL can't capture the cross domain information in inputs.

For pruning weights, numerous researches [51, 26, 8, 13, 11] have shown that removing a large portion of connections or neurons won't cause significant performance drop. Pruning algorithms often seek certain ways to introduce a criterion for evaluating the relative importance of channel, filter or individual weight. Then, such criterion is used for pruning, where least important weights can be pruned. Sparsity induced methods [43, 47, 28] can be regarded as a similar methods compared to pruning. In [43], group lasso are used as a regularization at training time. After the weights are close to zero, it can be safely pruned from the network. But using sparsity constraints often results in near zero solutions, some works [44] argue that small weights are in fact important for preserving performance. Some data-driven pruning methods [13] can avoid this problem by designing the criterion based on the intermediate feature maps. Other than data-driven approaches, certain optimization methods [47] for sparsity constraints can alleviate this problem too. From another perspective, the goal of pruning algorithm is to reduce the unique weights in the model and remove the others. Our method have the same goal here, however, we don't remove other weights, we make them share the same channel.

Besides weight pruning and sharing, other popular methods include matrix factorization [35], knowledge distillation [12, 45, 22], and variational inference approaches [29, 27].

2.2. Cross Domain Applications

In this paper, we focus on two types of cross domain tasks, the first one is two-stream action recognition, the second is RGB-D scene classification.

One of the most popular methods regarding action recognition is two-stream CNNs. After [37] proposed a method that uses RGB and stacked optical flow frames as appearance and motion information respectively, this kind of methods gets more and more attentions [42, 5]. Our cross domain compression framework is based on this series of works, because the architecture of these kind of methods are close to image classification task, which makes it possible to apply numerous compression methods on such methods. Another reason to choose two-stream methods is that RNN based algorithms [4] for action recognition rely on the features or outputs from corresponding RGB or flow CNN models and most memory usage and computation cost come from CNN part. Other action recognition methods like C3D [19], 3D-resnet [9] use 3-D convolution kernels to learn spatial and temporal information together. But existing compression techniques are harder to be applied on 3D CNN.

Scene classification is one of the basic problems in computer vision research. With cost affordable depth senor, Kinect, depth images can be used in scene classification task. Compared to RGB images, depth images can provide additional strong illumination and color invariant geometric cues. RGB and depth images fusion then become a promising way for scene classification. In this paper, we consider score level RGB-D fusion [6, 18], leave the intermediate feature maps untouched. RGB-D models are also suitable to apply compression techniques.

3. Method

In this section, we first show that previous weight sharing methods like Learning to Share [47] can't utilize underlying cross domain relationships. Then, we will introduce our method.

3.1. Learning to Share Revisit

In learning to share [47], they formulate the compression problem as a regularization problem. Group Lasso related methods have similar formulation, the regularization term is different. The formulation can be represented as:

$$\min_{\theta} \mathcal{L}(f_\theta(x)) + \mathcal{R}(\theta). \tag{1}$$

Here, in most classification task $\mathcal{L}$ is a cross entropy loss and $\mathcal{R}$ is the regularization term, f_θ is a neural network parameterized by θ. For learning to share, the regularization term is:

$$\mathcal{R}(\theta) = \sum_{l=1}^{L} \sum_{i=1}^{N_{l-1}} \lambda_{l,i} \|\theta_{l,i}\|, \tag{2}$$

where θ_l is the weight of lth layer, and $\theta_l \in R^{w_l \times h_l \times N_{l-1} \times N_l}$. w, h, N_{l-1}, N_l are the width, height, number of input channels and number of output channels in lth layer. The group is predefined along channel dimension. As we mentioned in section 2, $\sum_{i=1}^{N_{l-1}} \lambda_{li} \|\theta_{li}\|$ is a special regularization term called Group Ordered Weighted Lasso (GrOWL), which can force sparsity and learn underlying correlations among inputs at the same time.

A natural way to extend Learning to Share is to add GrOWL regularization to cross domain models. We only consider two domains in our experiments. To verify whether Learning to Share can learn cross domain correlation within the inputs, we designed a simple task. In this simple task, some modification are done on MNIST and two datasets MNIST-Rot (rotation by 45 degrees) and MNIST-Blur (motion blurred) are created as two toy domains. The weight correlation is calculated by:

$$S(i,j) = \frac{\theta_{l,i}^T \theta_{l,j}}{\|\theta_{l,i}\|_2 \|\theta_{l,j}\|_2}. \tag{3}$$

The feature correlation is calculated in Eq. 5.

We use LeNet-5 on each domain. GrOWL is applied except the first layer and the last layer. In Fig. 2, it clearly shows that after GrOWL regularization the correlation across weights from different domain model is close to zero, which indicates that GrOWL can't utilize underlying cross domain relationships.

Besides such drawback, hyper-parameter tuning is difficult and each layer has its own λ_l.

3.2. Cross Domain Task

To better explain our method, a formal definition of cross domain task is given. We use same network architecture on two domains except for the first layer, since inputs may have different number of channels. A typically DNN layer can be defined as a function parameterized by its weights, which can be expressed as: $y_l = f_{\theta_l}(x_l)$. Without loss of generality, the model in first domain can be defined as $y_{A,l} = f_{\theta_{A,l}}(x_{A,l})$. The second model can be defined by replacing A with B. Suppose the dataset D have m samples: $D = \{(x_{A_1}, x_{B_1}, y_1), \ldots, (x_{A_m}, x_{B_m}, y_m)\}$. Then the objective function has the form:

$$\min_{\theta_A, \theta_B} \mathcal{L}(f_{\theta_A}(x_{A_i}), f_{\theta_B}(x_{B_i})) + \mathcal{R}(\theta_A, \theta_B), \tag{4}$$

where $\mathcal{L}$ is cross domain task loss, and $\mathcal{R}$ is regularization loss.

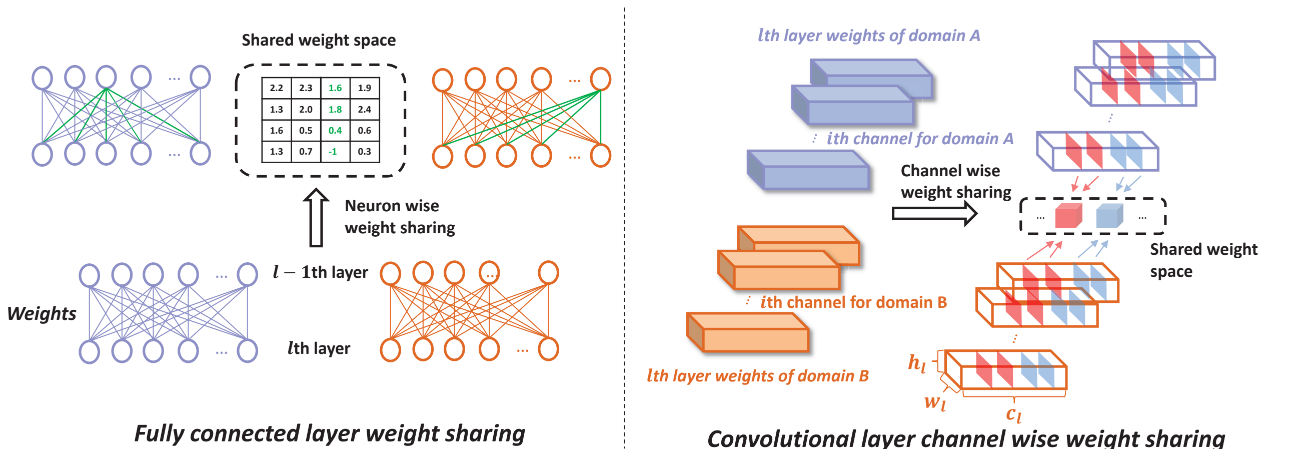

Shared weight space
2.2 2.3 1.6 1.9
1.3 2.0 1.8 2.4
1.6 0.5 0.4 0.6
1.3 0.7 -1 0.3
Neuron wise weight sharing
Weights
$l-1$th layer
lth layer
Fully connected layer weight sharing
lth layer weights of domain A
ith channel for domain A
Channel wise weight sharing
Shared weight space
ith channel for domain B
lth layer weights of domain B
h_l
w_l
c_l
Convolutional layer channel wise weight sharing

Figure 3: **Left figure** shows weight sharing in fully connected layers. **Right figure** shows weight sharing in convolution layers.

3.3. Graph Embedding as a Regularization

In section 3.1, we argue that Learning to Share is not sufficient for cross domain tasks not only because they can't discover cross domain correlation but also the hyper-parameter tuning is too time-consuming. Similar argument can be applied to Group Lasso method. When training model with GrOWL and Group Lasso, all weights in a layer often become zeros. If this happens, one have to adjust the hyper-parameter to train it again.

Hence, to solve these two problems, we aim to compress model by structured weight sharing. During training, the model is regularized with graph embedding constraint. After the model is fully trained, we cluster the weights according to transformed features. If we use fully shared approach, we won't suffer from the problem of training instability mentioned above. Fully shared approach won't turn all the weights in a layer to zero.

Algorithm 1 Graph Embedding Regularization

```
1: input: Middle layer output, $x_l^A$ or $x_l^B, l = 1, \dots, L$;
   Data set $D$ with $(x_i^A, x_i^B, y_i), i = 1, \dots, m$
2: initialization : $f_A$, $f_B$, $\mathcal{R}_{Spectral}$
3: for epoch $= 1$ to $N$
4:   $x_{t,l}^A = \text{Trim}(x_l^A)$
5:   $x_{t,l}^A = \text{Trim}(x_l^A)$
6:   $\mathcal{R}_{Spectral} = \mathcal{R}_{Spectral}(\text{concate}(x_{t,l}^A, x_{t,l}^B), \theta_s)$
7:   $\min_{\theta_A, \theta_B} \mathcal{L}(f_{\theta_A}(x_A), f_{\theta_B}(x_B)) + \mathcal{R}_{Spectral}$
8: end for
9: output: $f_A$, $f_B$, $R_{Spectral}$
```

Before introducing graph embedding constraint, we first show how we represent intermediate features. A naive way to represent similarity between input channels is to calculate the correlation between input features. Given an input of lth layer $x_l \in R^{W_l \times H_l \times C_l}$, W_l is the width of feature map and H_l is the height of feature map, C_l is the number of input channels. Suppose the number of data points in D is m, the inputs cross all samples can be represented as $X_l \in R^{W_l \times H_l \times C_l \times m}$, this X_l can be reshaped to a 2D representation $X_l^{2D} \in R^{C_l \times m W_l H_l}$. We can represent the similarity between input channels as below:

$$S_{x_l}(i,j) = \frac{X_l^{2D}(i,:)^T X_l^{2D}(j,:)}{\|X_l^{2D}(i,:)\|_2 \|X_l^{2D}(j,:)\|_2}. \tag{5}$$

In Eq.5, if input channel i and j is similar, then the inner product between x_l from all samples should be large too. However, the computation cost is expensive if x_l is large. For example, if l is the 13th layer of VGG-16, and we have 5×10^4 samples, then $X_l^{2D} \in R^{512, 49 \times 5 \times 10^4}$, each vector in X_l^{2D} will have 2.45 million dimensions. The computation cost will prohibit us to update input feature similarity matrix when training.

To make the update of input feature similarity matrix affordable, we apply average pooling to the feature map to reduce the size of it. If the feature map has size $W \times H \times C$, then the reduced feature map has size $w \times h \times C$, where wh is much less than WH. The size of feature map can be further reduced by random sample part of it. By doing so, the computation of similarity matrix is largely decreased and we call this operation Trim. For each input x_l, trimmed input feature map $x_{t,l}$ is:

$$x_{t,l} = Trim(x_l). \tag{6}$$

The similarity calculation is same in Eq. 5 with x_l replaced by $x_{t,l}$. During training, we replace m with the batch size

b for forward and backward calculation. After we have the similarity matrix between input channels of a layer, we try to cluster the weights according to the similarity map. Directly clustering weights on similarity map can result in performance drop. For this reason, graph embedding can be used as a constraint. Another reason we can use graph embedding [30, 41] is that it's well known for clustering on similarity graph which we already have.

Within the scope of graph embedding, similar formulation from SpectralNet [36], a recent proposed deep spectral clustering method, is used. Spectral clustering can be inserted into R in Eq. 3 and regularize the complexity of the model. In below, the detail of graph embedding regularization is given. As above mentioned, we use truncated input feature map to enable affordable intermediate layer similarity calculation. The intermediate similarity matrix calculation for cross domains uses Eq.5 by replacing $X_l^{2D}(i,:) = \text{concate}(X_{t,l}^{A,2D}(i,:), X_{t,l}^{B,2D}(i,:))$. 'concate' is a simple operation to join two vectors into one vector.

Then, the spectral clustering can be applied on intermediate similarity graph. Given a specific layer l, the graph embedding constraint has such form:

$$\mathcal{R}_{Spectral} = \sum_{l=1}^{L} \frac{1}{4C_l^2} \sum_{i,j=1:2C_l} \mathrm{S}_l(i,j) \| z_{l,i} - z_{l,j} \|_2^2, \quad (7)$$

where $\mathrm{S}_l \in R^{2C_l \times 2C_l}$ is the similarity matrix of lth layer inputs across two domains, C_l is the number of channels in input x_l^A or x_l^B. $z_l \in R^{2C_l \times k_l}$ is the output of spectral clustering, k_l is the target number of clusters for layer l. For spectral clustering, there is an additional constraint on z_l:

$$\frac{1}{2C_l} z_l^T z_l = \mathcal{I}. \quad (8)$$

And it requires to compute eigendecomposition on S_l to get z_l. However, the eigendecomposition is time-consuming to compute. Similar to SpectralNet, we use a neural network f_{sl} with a orthogonal layer to approximate the eigendecomposition. The orthogonal output is achieved by using Cholesky decomposition, interested readers can refer to Appendix B in [36]. By inserting f_{sl}:

$$z_{l,i} = f_{sl}(X_l^{2D}(i,:)). \quad (9)$$

As aforementioned, $X_l^{2D}(i,:) = \text{concate}(X_{t,l}^{A,2D}(i,:), X_{t,l}^{B,2D}(i,:))$. Simply use standard spectral clustering may cause the unbalance of clusters, which will limit the capacity of the model. Alternatively, we use normalized spectral clustering to impose balanced clusters.

$$R_{Spectral} = \sum_{l=1}^{L} \frac{1}{4C_l^2} \sum_{i,j=1:2C_l} \mathrm{S}_l(i,j) \| \frac{z_{l,i}}{d_i} - \frac{z_{l,j}}{d_j} \|_2^2, \quad (10)$$

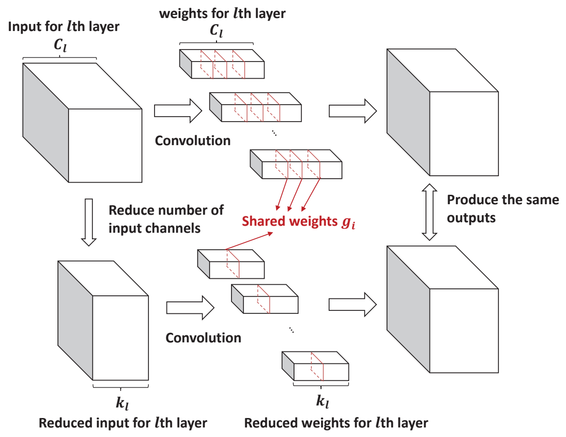


Figure 4: Illustration of how to reduce computation cost for our proposed method. It can be understood as reducing the number of input channels in feature maps and weights. Original and reduced version can both produce the same output.

where $d_i = \sum_i^{2C_l} \mathrm{S}_l(i,j)$. The final objective function for our method can be expressed as:

$$\min_{\theta_A, \theta_B} \mathcal{L}(\theta_A, \theta_B) + \mathcal{R}_{Spectral}, \quad (11)$$

where $\mathcal{R}_{Spectral}$ and $\mathcal{L}(\theta_A, \theta_B)$ are defined in Eq.10 and Eq.4 separately.

3.4. Weight Sharing

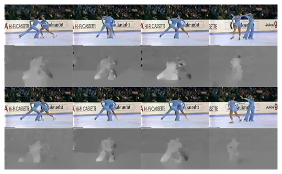
After training the model with objective function Eq. 11, we are ready to cluster the features according to the $z_l \in R^{2C_l \times k_l}$ for each layer. As normal spectral clustering process, we use K-means to cluster the features based on z_l. Since we have the clusters of features, it can be used to guide the clustering of weights. If channels i, j from intermediate features are in the same cluster, weights of channels i, j will also have the same cluster. The detailed sharing process is depicted in Fig. 3. Once weight clustering finished, we fine-tune the model according to clustering result. Suppose the ith group of weights in layer l have $n_{l,i}$ input channels, the weights in ith group is replaced by the centers $g_{l,i}$ of this group. The gradient computation of centers is:

$$\frac{\partial L}{\partial g_{l,i}} = \frac{1}{n_{l,i}} \sum_{\theta_{l,j} \in \mathcal{G}_{l,i}} \frac{\partial L}{\partial \theta_{l,j}}, \quad (12)$$

where $\mathcal{G}_{l,i}$ is the set containing all instances in this group.

3.5. Improve Inference Speed

In this work, we mainly focus on compressing the model instead of reduce computation cost, but we still can achieve


(a) RGB and optical flow frames

(b) RGB and depth images

Figure 5: Example of dataset images, (a) is the RGB and optical flow images within UCF-101 dataset, (b) is the RGB and depth images from SUN RGB-D dataset

moderate improvement concerning computation cost. It can be shown that we can reduce the number of channels by a fraction of $\frac{C_l}{k_l}$. Unlike WSNet, we don't need a special designed algorithm to reduce computation cost. The weight channels in the same group can be replaced by one channel, and corresponding feature maps can be replaced by one feature map averaging all feature maps in the group. Such replacement won't change the outputs. Details are shown in Fig. 4.

3.6. Benefit of Cross Domain Sharing

As we described above, one of the benefits of weight sharing is that it provides a natural way to speed up at inference time. Another advantage of cross domain weight sharing is it allows larger model capacity compared to any other single domain compression method. For a specific layer with input size n_{input}, if we want a $20\times$ compression rate, for single model compression, we can only keep 5% of the weights for each model, but for cross domain weight sharing, we can have $0.1n_{input}$ clusters, which is two times more than single domain compression method. Notice that weight sharing is key to achieve such result. The relative larger model capacity is especially important if required compression rate is extreme.

4. Experiment

We assess the proposed method on three different datasets with two tasks. We compare our method with a series of pruning and sparsity induced methods. The pruning algorithms including structured weight pruning [13, 26] and individual weight pruning [51, 8]. The reason why we only compare pruning and sparsity induced method is that these methods are the majority of model compression algorithms. Moreover, quantization methods focus on single weight value sharing and can be applied on the basis of pruning algorithms and the proposed method.

4.1. Implementation Details

Our method and related comparison methods are all implemented in pytorch [32], some of the comparison methods are based on the implementation of [52]. Sparsity induced methods are only applied on scene classification task, since in action recognition task, we can't find suitable hyper-parameters for GrOWL or Group Lasso, some of layers always become zero whether we use proximal gradient or soft-thresholding as optimization method.

For SUN-RGBD dataset, we train model with graph embedding constraint for 100 epochs with batch size of 128. SGD with momentum is used as optimizer, momentum is set to 0.9 and start learning rate is 0.03. Learning rate is decayed by a factor of 0.1 for every 30 epochs. After training completely, weights sharing are performed as described in section 3.4. In weight sharing stage, the model is fine-tuned for 60 epochs with the same optimizer and learning rate is set as 3×10^{-3} with the same scheduler.

For action recognition dataset, the models are trained on each domain separately with the settings in [42] and five-crops data augmentation. The models are put together and trained with graph embedding constraints for 80 epochs with SGD and momentum 0.9, the start learning rate is 1×10^{-4} and batch size is 32. After clustering, models are fine-tuned with the same learning rate for 60 epochs.

4.2. Datasets

SUN-RGBD Dataset [39] contains 10,355 RGB and Depth image pairs captured from different cameras. We follow the experimental settings in [18]. 19 categories are kept for our experiments with 4,845 images for training and 4,659 images for testing.

UCF-101 Dataset [40] comprises of realistic videos collected from Youtube. It contains 101 action categories, with 13,320 videos in total (9,537 videos for training, the rest for testing). UCF-101 split-1 is used for training and testing.

HMDB-51 Dataset [24] contains a total of about 7,000 video clips distributed in a large set of 51 action categories. Each category contains a minimum of 101 video clips. We use split-1 in official release of HMDB-51 dataset.

4.3. RGB-D Scene Classification

For SUN-RGBD dataset. we follow the same experiment setting in [18]. HHA images are extracted follows [6]. As we discussed in Section 3, we calculate average score fusion across two domains. Also class weighted cross entropy is used as a common practice, the weight for each class is given by $w(t) = \frac{N_{c_{max}} - N_{c_{min}}}{N_t - N_{c_{min}} + \tau}$, where $N(t)$ is the number of examples in tth class, c_{max} is the class with most samples, c_{min} is the class with least samples. For both domains, we use AlexNet pre-trained on Placed365 dataset [48].

In Table 1, we list the network settings for Sun RGB-D dataset. k_A and k_B are two different settings for our method. The setting for GrOWL is the result after training with GrOWL regularization. The number in the list is the unique input channels for cross domain models.

Table 1: Network settings for AlexNet [23] on SUN RGB-D dataset.

Layer	original	k_A	k_B	GrOWL
conv1	6	6	6	6
conv2	128	32	16	12
conv3	384	96	48	12
conv4	784	192	96	21
conv5	512	128	64	72
fc1	18432	1024	512	1037
fc2	8192	512	512	423
fc3	8192	8192	8192	8192

Table 2: Results of SUN RGB-D dataset.

Method	Performance	Rate
Original	47.32%	1
GrOWL [47]	44.28%	17.6
Ours k_A	47.21%	14.8
Ours k_B	**47.01%**	22.8

In Table 2, it can be shown that the performance of GrOWL is lower than our proposed method by near 3%. Even though the compression rate of GrOWL is similar to setting k_B of our method. This shows that, for cross domain models, sparsity induced method usually gives suboptimal solutions for cross domain compression. Furthermore, our method can be regarded as GrOWL without sparsity. In this experiment, we give two settings k_A and k_B for our method. Though, the compression rate is variant, only little difference is observed for performance, which shows that our method is robust against hyper-parameter tuning. On the other hand, GrOWL is sensitive to hyper-parameter, the result in Table 2 is achieved by more than ten rounds of experiments given different hyper-parameter settings in GrOWL.

4.4. Action Recognition Dataset

For action recognition tasks, during training we combine two popular methods TSN [42] and two-stream [5]. VGG-16 is used for action recognition task. As in [5], we use 5-crops data augmentation in training. The optical flow images are extracted based on [46]. Following TSN, we split a video into three segments, and random samples RGB frame for each segment. Once we have the index of RGB frame, we sample the same index and following 10 frames in horizontal and vertical optical optical flow. The horizontal and vertical flow images are stacked to a $224 \times 224 \times 20$ cubic to feed into optical flow DNN model.

For our method, we set hyperparameter $k_l = \frac{2C_l}{r}$. r is set to 2, 4 or 8 for different settings. For a relative fair comparison we set the pruning rate (p-rate in Table 4) equal to 0.3, 0.5 or 0.75 separately.

Table 3: Network settings for VGG-16 [38] of action recognition dataset for proposed method

Layer	original	k_A	k_B	k_C
conv1	23	23	23	23
conv2,3	128	32	32	16
conv4,5	256	64	64	16
conv6 to 8	512	128	128	64
conv9 to 13	1024	256	128	64
fc1	50176	1024	512	256
fc2	8192	512	512	256
fc3	8192	8192	8192	8192

Table 4: Network settings for VGG-16 [38] on action recognition dataset for comparison methods.

Layer	original	p-rate 0.3	p-rate 0.5	p-rate 0.75
conv1	3	3	3	3
conv2,3	64	44	32	16
conv4,5	128	90	64	32
conv6 to 8	256	180	128	64
conv9 to 13	512	358	256	128
fc1	25088	17561	12544	6272
fc2	4096	2867	2048	1024
fc3	4096	4096	4096	4096

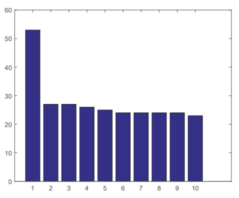
In Tables 3 and 4 we list the detail of target network structure of our method and comparison methods. The major difference between Table 3 and Table 4 is that in Table 3, all the settings are for both domains, on the contrary, 4 are only for single domain. For example, in conv2 of k_A, we have 32 unique channels for 128 channels in both RGB and optical flow models. In conv2 of p-rate 0.5, 32 is also given here, this is only for RGB or optical flow model, for both models, at p-rate 0.5, there are 64 unique channels in weight matrix. Table 5 shows the results for UCF-101 dataset and HMDB-51 dataset. The number following comparison methods is the pruning rate (p-rate) for the method. For example, 'prune or not prune 0.5' means prune or not prune method at pruning rate of 0.5. Clearly, our method can achieve the best results (trade off between performance and compression rate) compared two all the other methods. Morever, individual weight pruning algorithms is significant better than group weight pruning algorithms (almost 10% absolute improvement). Group weight sharing methods like Apoz [13] and Efficient Network [26] often suffer from large performance drop (6% to 10% compared to original) even only with a small fraction of pruning-rate

Table 5: Overall results for action recognition dataset

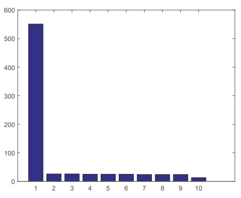
Method	Performance	Rate
UCF-101 Dataset		
Original	88.52%	1
Prune or not prune 0.5 [51]	87.7%	2
Sensitity [8] 0.5	87.9%	2
Efficient convnet[26] 0.5	78.3%	2
Apoz 0.5 [13]	79.6%	2
Prune or not prune 0.75 [51]	83.8%	4
Sensitity 0.75[8]	77.9%	4
Efficient convnet [26] 0.75	58.9%	4
Apoz 0.75 [13]	69.6%	4
Ours k_A	88.21%	12
Ours k_B	88.9%	23
Ours k_C	87.7%	46
Original 5-crops	90.8%	1
Ours k_B 5-crops	**91%**	**23**
HMDB51 Dataset		
Original	57.51%	1
Apoz 0.3 [13]	53.6%	1.42
Efficient convnet 0.3 [26]	51.8%	1.42
Apoz 0.5 [13]	47.7%	2
Efficient convnet 0.5 [26]	20.8%	2
Ours k_B	57.4%	23
Ours k_C	56.9%	46
Original 5-crops	59.9%	1
Ours k_B 5-crops	**59.8%**	**23**

(0.3 or 0.5). These observations are inconsistent with single RGB model pruning results. At least, at pruning rate 0.3 or 0.5, many algorithms can maintain the performance. There might be many reasons for this phenomena, the model capacity required for non-RGB domain might be larger than RGB domain, thus, pruning some channels may hurt the performance severely. Another possibility is the difficulty of the dataset, HMDB-51 is believed to be more difficult than UCF-101. As a result, it's not easy to keep performance on HMDB-51 dataset.

Another interesting phenomena is that our method are robust to a set of different hyperparameters. The performance starts to drop (less than 1% absolute performance lost) after a relative high compression rate (46 times). For 5 different settings across three datasets, the largest difference before and after compression is 0.8%. In setting k_B of UCF-101, our method is better than original by 0.4%. Overall speaking, our method is much easier for hyperparameter searching compared to sparsity induced method, and it can achieve better trade-off compared to pruning algorithms.



(a) setting A



(b) simple constraint

Figure 6: Group size of largest 10 groups in layer conv13 of VGG-16. Setting A, in figure (a), can achieve 88.2%. Simple constraint in figure (b) can achieve 87.5%. Random group can achieve 87.3%.

4.5. Study of group size

Our method are further compared with random sharing and simple similarity constraint. Naively, given a similarity map Sim_l at layer l, we define:

$$R_s = \begin{cases} 1 - S_l(i,j), & \text{if } S_l(i,j) \le t, \\ S_l(i,j) & \text{otherwise.} \end{cases} \quad (13)$$

This indicates that we push the feature maps and weights closer if their channel similarity is greater than t. t is set as 0.3. Using such constraint will result in highly unbalanced group in the compressed model. From Fig. 6, it is obvious that large and unbalanced group hurt the performance and make the results close to random sharing. This shows that one key ingredient for our method is to have balanced groups.

There are some groups with group size 1 in both domains. This can be regarded as domain private parts which only captures domain specific information. In domain separation networks [2], one can find similar arguments. Resulting compressed model can be separate into two parts, domain common parts and domain separate parts. Following this argument, our method can be viewed as an approach to identify domain common part within cross domain models. Domain common part is essential for cross domain model compression, since it can be reused across different domain.

5. Conclusion

In this paper, we solve the problem of model compression in cross domain settings. To achieve such goal, we use graph embedding as a regularization for cross domain models. The weights are structurally shared according to the results of clustered features. Our method can achieve the state of the art result on compression rate with little performance loss on two different tasks. Group size within each layer is identified to be one of the key elements to the success of our method.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. Domain separation networks. In *Advances in Neural Information Processing Systems*, pages 343–351, 2016.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[4] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[5] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1933–1941, 2016.

[6] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision*, pages 345–360. Springer, 2014.

[7] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[8] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[9] K. Hara, H. Kataoka, and Y. Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. In *Proceedings of the ICCV Workshop on Action, Gesture, and Emotion Recognition*, volume 2, page 4, 2017.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[11] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[13] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[14] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[15] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[17] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[18] Z. JG, H. KQ, et al. Df2net: A discriminative feature learning and fusion network for rgb-d indoor scene classification. 2018.

[19] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[20] Q.-Y. Jiang. Deep cross-modal hashing. In *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[21] X. Jin, Y. Yang, N. Xu, J. Yang, N. Jojic, J. Feng, and S. Yan. Wsnet: Compact and efficient networks through weight sampling. In *International Conference on Machine Learning*, pages 2357–2366, 2018.

[22] J. Kim, S. Park, and N. Kwak. Paraphrasing complex network: Network compression via factor transfer. *arXiv preprint arXiv:1802.04977*, 2018.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[24] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2556–2563. IEEE, 2011.

[25] C. Li, C. Deng, N. Li, W. Liu, X. Gao, and D. Tao. Self-supervised adversarial hashing networks for cross-modal retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4242–4251, 2018.

[26] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[27] C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[28] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[29] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[30] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[31] U. Oswal, C. Cox, M. Lambon-Ralph, T. Rogers, and R. Nowak. Representational similarity learning with applica-

tion to brain networks. In *International Conference on Machine Learning*, pages 1041–1049, 2016.

[32] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017.

[33] A. Polino, R. Pascanu, and D. Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[34] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[35] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013.

[36] U. Shaham, K. Stanton, H. Li, B. Nadler, R. Basri, and Y. Kluger. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*, 2018.

[37] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[38] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[39] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[40] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[41] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[42] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision*, pages 20–36. Springer, 2016.

[43] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[44] J. Ye, X. Lu, Z. Lin, and J. Z. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. *arXiv preprint arXiv:1802.00124*, 2018.

[45] J. Yim, D. Joo, J. Bae, and J. Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2017.

[46] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pages 214–223. Springer, 2007.

[47] D. Zhang, H. Wang, M. Figueiredo, and L. Balzano. Learning to share: Simultaneous parameter tying and sparsification in deep learning. 2018.

[48] B. Zhou, A. Khosla, A. Lapedriza, A. Torralba, and A. Oliva. Places: An image database for deep scene understanding. *arXiv preprint arXiv:1610.02055*, 2016.

[49] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[50] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[51] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[52] N. Zmora, G. Jacob, and G. Novik. Neural network distiller, June 2018.